Funkce

Př. Výpočet kombinace čísel .

$$C = \frac{n!}{k!(n-k)!}$$

```
program  kombinace;
var n, k : integer;
      nf,kf,nkf : real;

procedure faktorial(var x:real, n: integer);
var i,c: integer
if (n>0) then begin
                        x=1;
                        for i:=n downto 1 do x:=x*i;
                  end
          else writeln('neni definovan');
end;
begin
write('Zadej n, k');
read(n,k);
faktorial( nf, n );
faktorial( kf, k );
faktorial( nkf, n-k );
c=nf/(kf*nkf);
writeln(' pocet kombinaci=', c);
end.
```

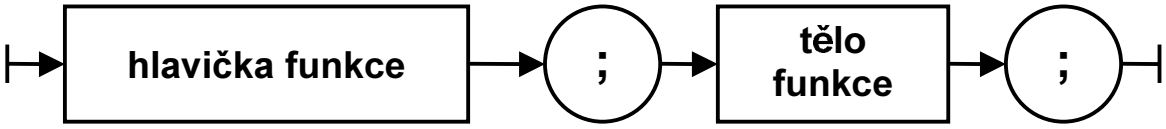
V některých případech je potřeba použít výsledné hodnoty vypočtené v proceduře jako operand ve výrazu. Volání procedury se však ve výrazu vyskytovat

nesmí ⟶ nejprve musíme vyvolat proceduru a výsledky pak použít ve výrazu.

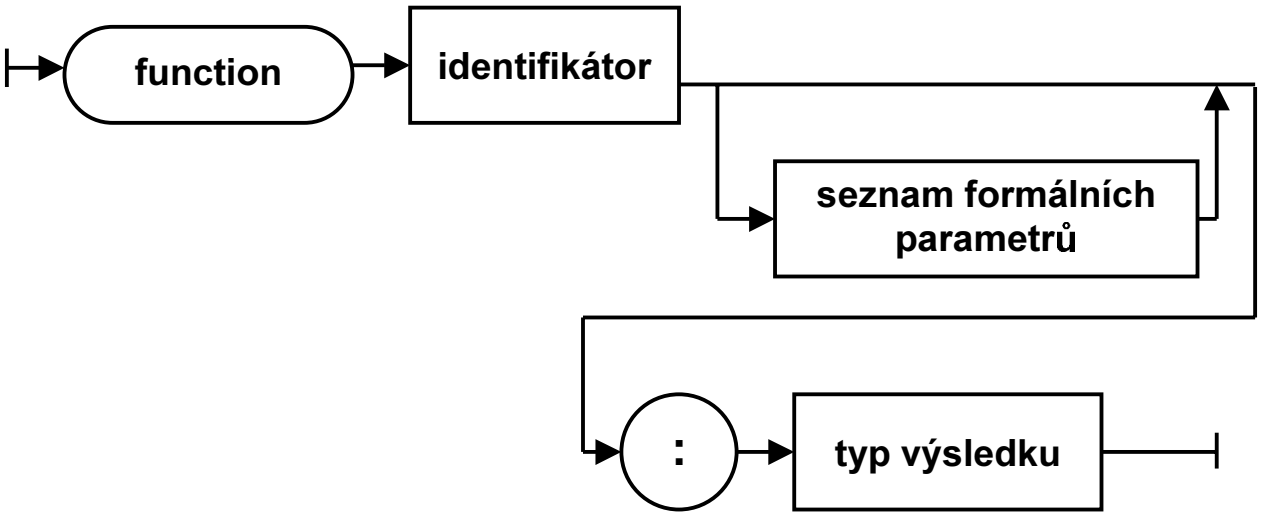
Tuto nevýhodu odstraňují uživatelsky definované *funkce.*

Deklarace funkcí v PASCALU

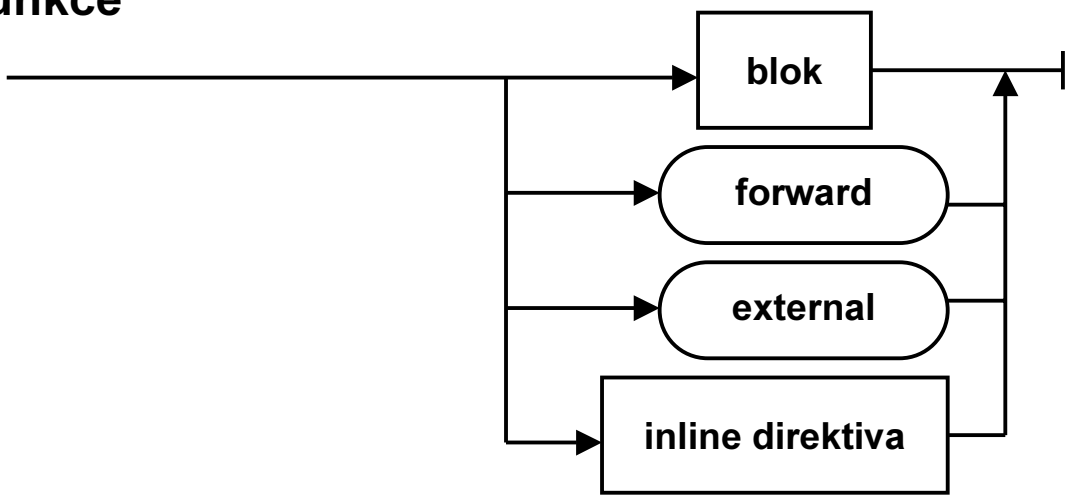
deklarace funkce



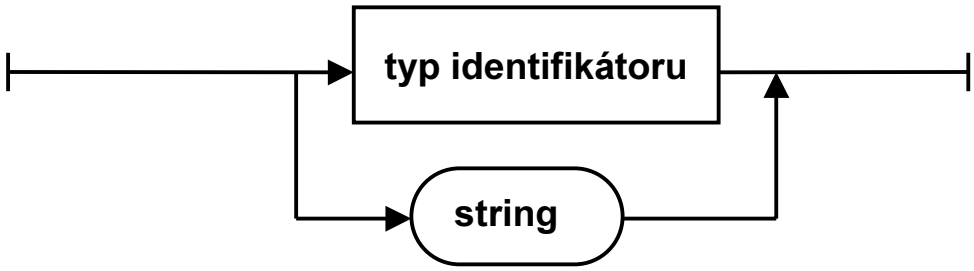
hlavička funkce



tělo funkce



typ výsledku



Př.

Deklarace funkce:

```
function F1( formální parametry ) : typ_výsledku;
  …
    lokální deklarace
  …
begin
  …
    tělo funkce;
  …
    F1:= výraz;
  …
end;
```

Volání funkce:

```
Var X: typ_výsledku

Begin
…
X:=F1(skutečné parametry);
…
end.
```

Vše co platí pro procedury (předávání parametrů, typy parametrů, lokální deklarace apod.) platí také pro funkce.

Kromě toho platí následující pravidla:

- **Funkce vrací hodnotu a nemůže tedy být použita jako samostatný příkaz, ale pouze jako součást výrazu (nejčastěji na pravé straně přiřazovacího příkazu).**

- **Funkce vrací pouze hodnotu jednoduchého typu a typu string !!!!!!!!!! Chceme-li, aby funkce vracela strukturovaný typ je nutné předat strukturovaný typ prostřednictvím parametrů volaných odkazem.**

- **Uvnitř těla funkce se musí objevit následující přiřazení**

  Identifikátor := výraz,

  kde *identifikátor* je identifikátor procedury, výraz musí být kompatibilní vzhledem k přiřazení s typem, který funkce vrací.

Př. Výpočet kombinace čísel .

```
program  kombinace;
var n, k : integer;
      c : real;

function faktorial( n: integer ) : real ;
var i : integer;
      vysl :real;
if (n>0) then begin
                           x=1;
                            for i:=n downto 1 do x:=x*i;
                            faktorial := x;
                  end
          else faktorial := 0;
end;
begin
write('Zadej n, k');
read(n,k);
c=faktorial(n)/(faktorial(k)*faktorial(n-k));
writeln(' pocet kombinaci=', c);
end.
```

POZOR !!!
Ve funkci je možné používat parametry volané odkazem, tj. funkce má kromě hlavní návratové hodnoty ještě vedlejší výsledek uložený do proměnné volané odkazem (tzv. vedlejší efekt funkce) → tento efekt může způsobit v některých případech chybné vyhodnocení výrazu.

Př.

```
function FCE(var X: integer ) : integer;
 begin
   X:=X+1;
   FCE := X*X;
 end;


begin
…

A:=1;
B:=A+FCE( A );
Write(A,B);
…
end.
```

Na obrazovku se vypíše buď 2 5 nebo 2 6 v závislosti na tom, zda překladač vyhodnotí výraz v pořadí A,FCE(A),+ nebo v pořadí FCE(A),A,+ (ve druhém případě bude použita funkcí upravená hodnota).

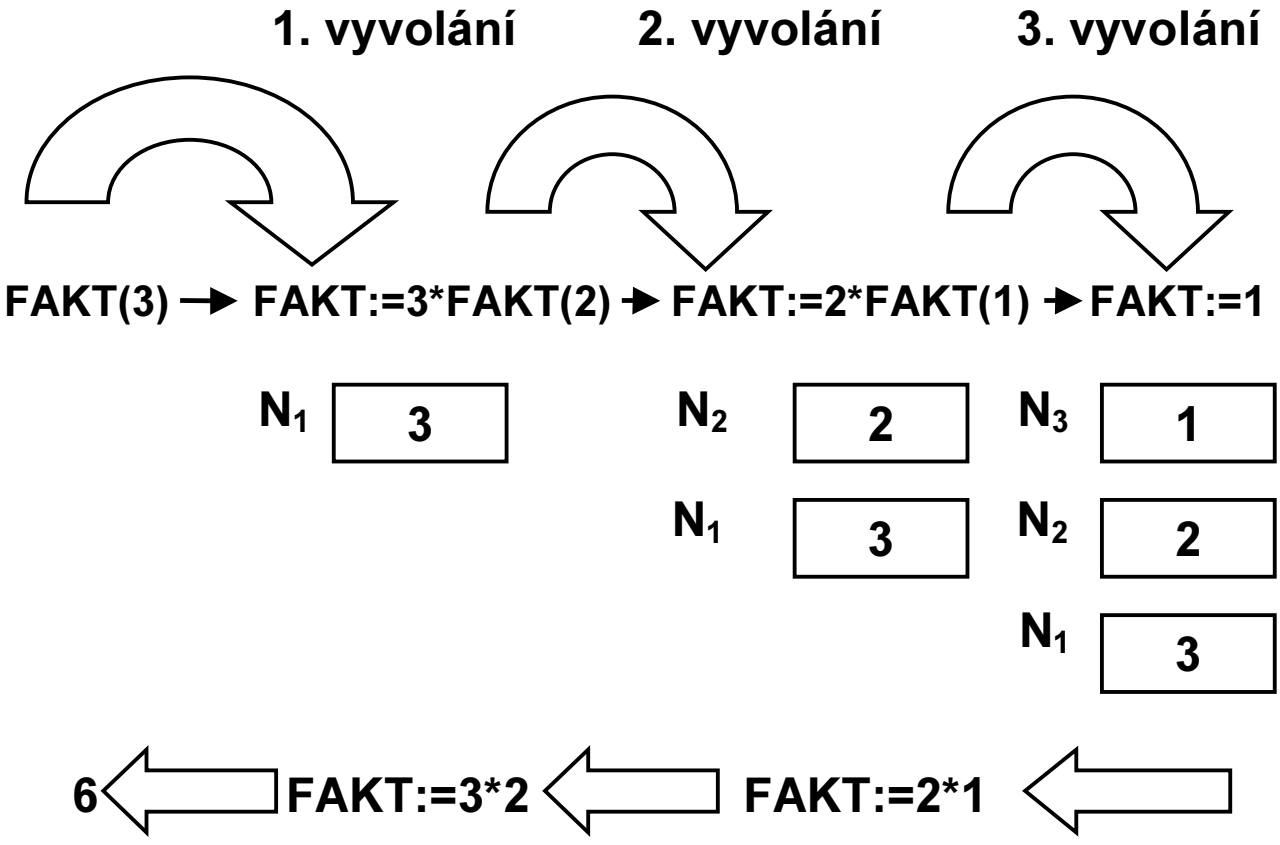
Rekurzivní procedury a funkce

Objekt je rekurzivní, je-li definován pomocí sama sebe.

Procedura (funkce) je rekurzivní pokud má v těle procedury (funkce) obsaženo volání sebe sama.

Př.

```
function FAKT(n:integer) : real
begin
 if n=0 then FAKT := 1 else FAKT:=n*FAKT(n-1);
end;
```



Rekurze :

- **Přímá - v proceduře (funkci) P se vyskytuje procedury (funkce P).**
- **Nepřímá - procedura P volá proceduru Q a ta volá přímo, popř. nepřímo proceduru (funkci) P.**

Rekurzivní procedury (funkce):

Použití: zápis algoritmů, které jsou definovány rekurzivně (faktoriál, nekteré algoritmy řazení polí, práce s dynamickými dat. strukturamy např. seznam strom atd.)

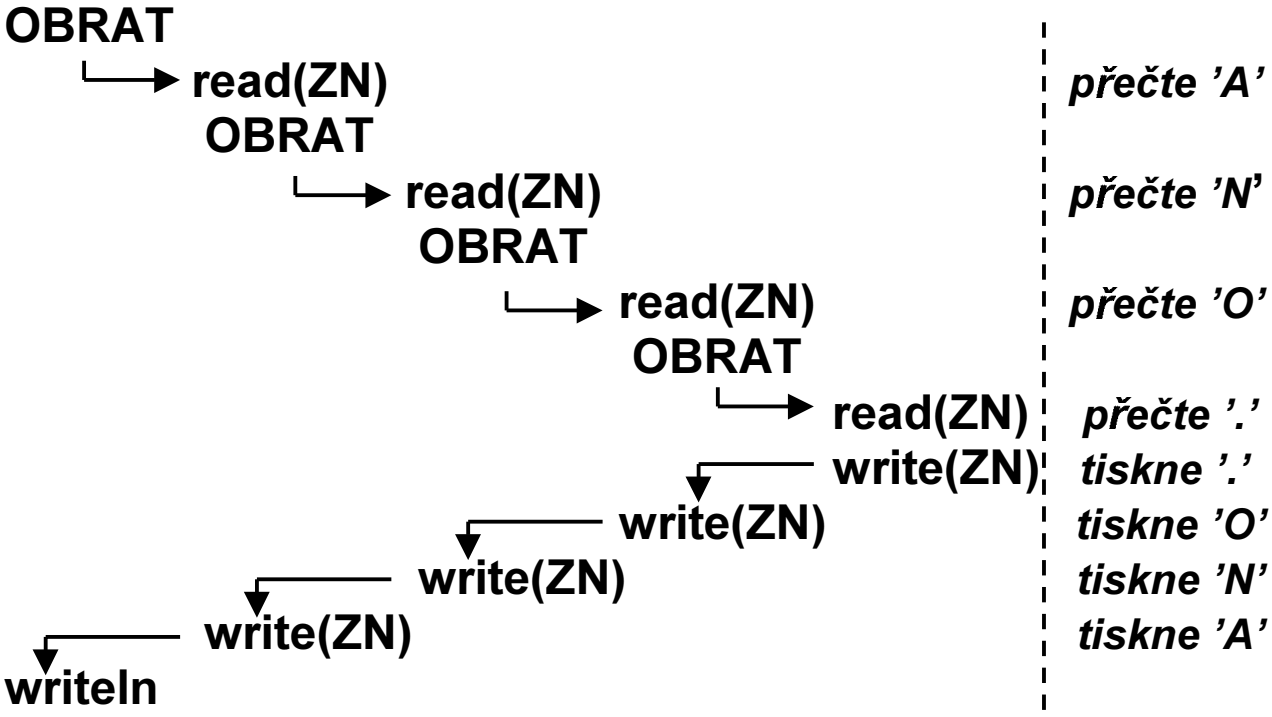
Výhody : stručný a přehledný zápis

Nevýhody : paměťově a časově náročnější než nerekurzivní verze (při každém vyvolání vznikají nové lokální proměnné a provádí se předávání parametrů)

Př. Rekurzivní procedura

```
program obraceni_vety;
  procedure OBRAT;
    var ZN: char ;
    begin
      read(ZN);
      if Z<>'.' Then OBRAT;
      write(ZN);
    end;
begin
 OBRAT; writeln
end.
```

Vstup : ANO.



Zásady pro psaní rekurzivních procedur (funkcí):

1. **Příkazová část rekurzivní procedury (funkce) musí vždy obsahovat větev pro „ triviální alternativu“, tzn. alternativu pro nějakou základní hodnotu parametru nebo vstupních dat , v níž nedojde k novému (rekurzivnímu) vyvolání procedury (funkce)**

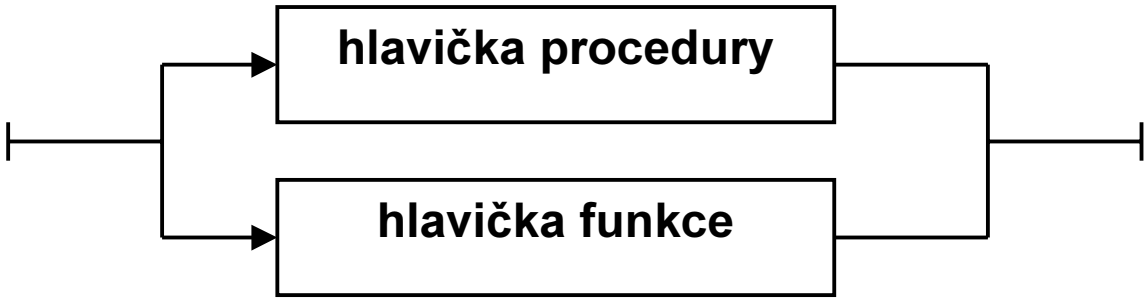
2. **Rekurzivní volání procedury (funkce) uvnitř příkazové části musí být provedeno s vhodně upraveným parametrem (nebo globálním objektem), přičemž tato úprava musí zajišťovat dosažení triviální alternativy po konečném počtu opakování.**

Poznámka: **Rekurzivní problémy je možné přepsat jako nerekurzivní, v některých případech je však nutné použít speciálních datových struktur (zásobník).**

Procedurální typy

Turbo Pascal dovoluje pracovat s procedurami (funkcemi) jako s objekty, které mohou být přiřazeny do proměnných.

Deklarace procedurálních typů :




```
type   Proc =    procedure;
       SwapProc = procedure (var X,Y: integer);
       MathFunc = function (X : real) : real
```

Jména parametrů mohou být libovolná, nemají žádný vliv na vlastní deklaraci procedury (při vlastní deklaraci procedury mohou být použity jiné identifikátory).

Procedurální proměnné - deklarace:

```
Var P : SwapProc;
    F : MathFunc;
```

Použití procedurálních proměnných :

```
{$F+}

procedure Swap( var A, B : integer )
 var POM : integer;
 begin
   POM := A; A := B; B := POM;
 end;

function Tan( Angle : real) : real;
 begin
  Tan:= sin(Angle) / cos(Angle);
 end;

{$F-}
```

Proměnným P a F mohou být přiřazeny hodnoty :

```
P:= Swap;
F:= Tan ;
```

pak volání

P(I,J)	je ekvivalentní s	*Swap(I,J);*
A:= F(X)	je ekvivalentní s	*A:= tan(X);*

Procedurální proměnné může být přiřazen identifikátor procedury, která je s ní kompatibilní vzhledem k přiřazení, tj. splňuje následující vlastnosti:

- **procedura musí mít stejný počet parametrů jako je počet parametrů uvedených v deklaraci procedurální proměnné**
- **parametry v odpovídajících pozicích musí mít identický typ**
- **nesmí to být standardní procedura nebo funkce**
- **nesmí to být vnořená procedura nebo funkce**
- **nesmí to být *inline* popř. *interrupt* procedura nebo funkce**
- **kompilace musí být provedena s direktivou {$F+}**

Procedurální parametry procedur a funkcí

- **jsou užitečné v situacích, kdy je jistá společná akce prováděna množinou procedur, popř. funkcí.**

Při deklaraci a použití procedurálních parametrů platí stejné zásady jako pro deklaraci procedurální proměnné.

Př.
```
program tabulky;
 type  Func=function(X,Y : integer) : integer;
 {$F+}
 function soucet(X,Y : integer) : integer;
   begin
     soucet := X + Y;
   end;
 function soucin(X,Y:integer) : integer;
    begin
      soucin:=X*Y;
    end;
 {$F-}
 procedure TiskTab(W,H:integer; OPER : Func);
  var X,Y : integer;
 begin
   for Y:=1 to H do
    begin
      for X:=1 to W do write(OPER(X,Y):5);
      writeln;
    end;
  writeln;
 end;
begin
 TiskTab(10,10,Soucet);
 TiskTab(10,10,Soucin);
end.
```

```
1    2    3    4    5    6    7    8    9    10
2    4    6    8   10   12   14   16   18    20
3    6    9   12   15   18   21   24   27    30
4    8   12   16   20   24   28   32   36    40
  …
  …
```

Datový typ soubor

Údaje se kterými počítač pracuje jsou uchovávány v operační paměti. Operační paměť potřebuje ke své činnosti zdroj energie – po vypnutí počítače se data z paměti počítače ztrácí → potřeba uchovávat data na energeticky nezávislých vnějších paměťových médiích (magnetické disky, pásky).

Na vnějších médiích se data uchovávají ve formě souborů tj. posloupnosti libovolných složek stejného typu. Soubor si lze zjednodušeně představit jako pole záznamů uložených na disku popř. pásce.

Zpracování dat v souborech – postupné (sekvenční).

Soubory v Turbo Pascalu - odlišné oproti standardnímu Pascalu

- **soubory s udaným typem (binární, datové soubory)**
- **textové soubory**
- **soubory bez udaného typu**

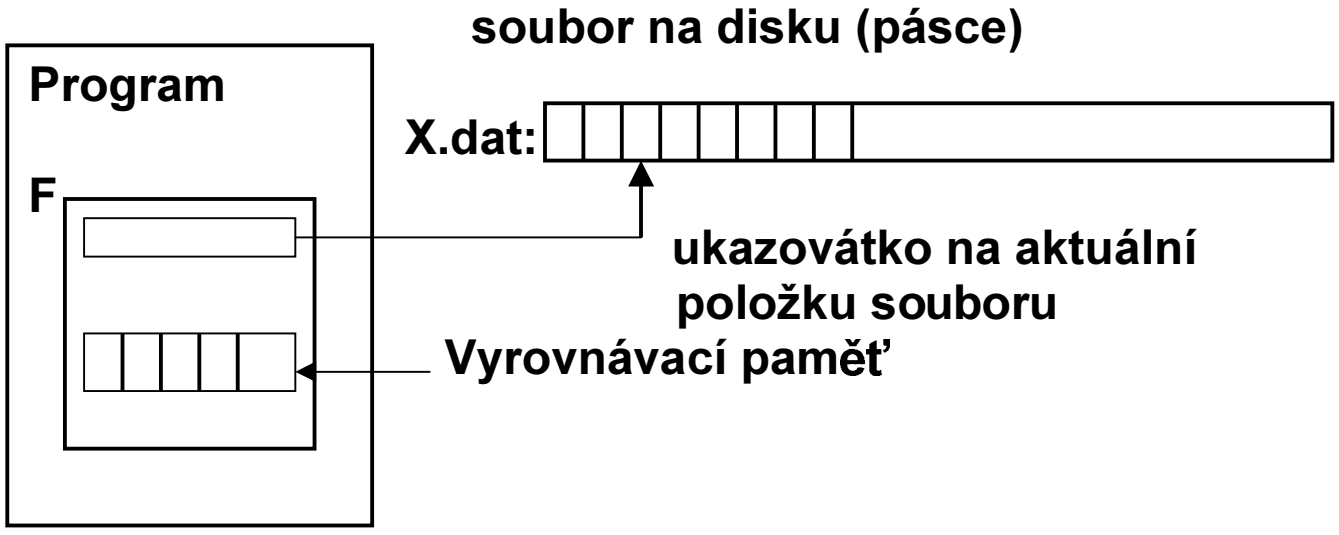
Operace s datovým typem soubor

Soubory nejsou objekty umístěné v operační paměti

Všechny odkazy na soubor jsou prováděny pomocí vstupních / výstupních operací, které jsou přístupné pomocí knihovních procedur a funkcí. Je-li soubor předáván jako parametr procedury popř. funkce musí být vždy volán odkazem !!!!

Základní kroky důležité při práci se souborem:

1. **deklarace proměnné typu soubor (vytvoření virtuálního souboru v paměti)**
2. **vytvoření vazby mezi virtuálním souborem a fyzickým souborem na disku (pásce)**
3. **otevření souboru (vyhledání souboru na disku (pásce) a nastavení odpovídajících ukazatelů na položky souboru)**
4. **práce se souborem (zápis dat do souboru, čtení dat ze souboru).**
5. **uzavření souboru**



Soubory s udaným typem
(binární, datové soubory)

1. **Deklarace:**

   type soubor = file of *typ_složky*;

   var f : soubor;

   nebo

   var f : file of *typ_složky*;

***typ_složky* - libovolný jednoduchý popř. strukturovaný datový typ kromě typů *file*, *file of* , *text* (nelze vytvářet soubor souborů) !!!!**

f - identifikátor virtuálního souboru (v programu pracujeme pouze s tímto identifikátorem, ne se jménem fyzického souboru)

2. **Vytvoření vazby mezi fyzickým a virtuálním souborem**

   Assign(f,'*jméno_souboru*');

   ***jméno_souboru* - jméno fyzického souboru (pokus.dat). Pod tímto jménem je soubor uložen na disku (pásce)**

3. Otevření souboru

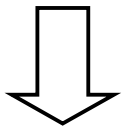
Reset(*F*)

Standardní procedura, otevírá soubor *F* pro čtení. Soubor musí existovat, jinak dojde k chybě (Run time error) !!!
Soubor se otevírá na začátku - aktuální se stává první položka

Rewrite(*F*)

Standardní procedura, otevírá soubor *F* pro zápis (přepis).
Pokud soubor neexistuje, tak se vytvoří nový soubor, jinak se soubor otevírá na začátku - aktuální je první položka, ostatní položky nejsou přístupné.

U otevřených souborů je vždy aktivní pouze jediná položka. Po provedení operací *read*, *write* se automaticky stává aktivní položka následující - soubor je zpracováván sekvenčně.



U sekvenčních souborů je oddělena fáze vytváření souboru (zápis položek na konec souboru) od fáze čtení / prohledávání souboru.

U souborů na discích je možné provádět při čtení tzv. přímý přístup (viz procedura *seek*).

4. **Práce se souborem - lze pracovat pouze s otevřeným souborem !!!!**

   eof(F) **- logická funkce, vrací *true* pokud se nacházíme na konci souboru (neexistuje další složka)**

   read(F,X) **- procedura přečte hodnotu aktivní složky souboru do proměnné X, pokud neexistuje další složka pak *eof(F)* má hodnotu true.**

write(F,X) **- procedura zapíše obsah proměnné *X* do souboru. Je-li soubor ve stavu zápis, je na konci souboru vytvořena nová složka, do níž je zapsána hodnota proměnné *X.* Je-li soubor ve stavu čtení, je hodnota zapsána do aktivní složky. Je-li nastavena aktivní složka o jedničku větší než pořadí poslední složky, je na konci vytvořena nová složka do které se vloží obsah proměnné *X* a *eof(F)* získá hodnotu *true*.**

seek(F,N) **- nastavení aktivní složky souboru F na pozici N (N : longint). Složky souboru jsou číslovány od počátku souboru počínaje nulou.**

filesize(F) **- funkce typu *longint,* jejíž hodnota udává počet složek souboru *F***

filepos(F) **- funkce typu longint, jejíž hodnota udává číslo aktivní složky souboru *F*.**

5. Uzavření souboru

Close(F) - zápis obsahu vyrovnávací paměti do souboru na disku

---

Př.

```
program datab;
 type datum=record
       den: 1..31;
       mesic: 1..12;
       rok: 1900..2000;
       end;
 type osoba=record
        jmeno, prijmeni: string;
        dat_nar: datum;
       end;
 var kartot:array [1..100] of osoba;
     i,n:integer;
     f: file of osoba;
 begin
  assign(f,'kart.dat');
  rewrite(f);
  writeln;
  write('Zadej pocet osob: '); readln(n);
  for i:=1 to n do begin
   with kartot[i] do begin
    write('jmeno: '); readln(jmeno);
    write('prijmeni: ');readln(prijmeni);
    write('den: '); readln(dat_nar.den);
    write('mesic: '); readln(dat_nar.mesic);
    write('rok: '); readln(dat_nar.rok);
    end;
    write(f,kartot[i]);
   end;
  close(f);
  end.
```

```
program datab;
 type datum=record
       den: 1..31;
       mesic: 1..12;
       rok: 1900..2000;
       end;
 type osoba=record
        jmeno, prijmeni: string;
        dat_nar: datum;
       end;
 var kartot:array [1..100] of osoba;
     i,n:integer;
     f: file of osoba;
 begin
  assign(f,'kart.dat');
  reset(f);
  writeln;
  i:=1;
  while ( NOT eof(f)) do begin
   read(f,kartot[i]);
   with kartot[i] do begin
    writeln('jmeno: ',jmeno);
    writeln('prijmeni: ',prijmeni);
    writeln('den: ',dat_nar.den);
    writeln('mesic: ',dat_nar.mesic);
    writeln('rok: ',dat_nar.rok);
    end;
   i:=i+1
   end;
  close(f);
  end.
```